Northeastern University - Seattle

Khoury College of Computer Sciences

CS6200 Information Retrieval Fall 2019

Lecture 8: Retrieval Models 1

Oct 21, 2019

Administrivia

- Assignment 2: feedback
  - When did you start on it?
- Quiz 6
  - Deadline Thursday 24th 9am, solutions next Monday
- Quiz 7 will be released today
  - deadline & solutions both next Monday
- Recording failure Oct 17th
  - Had problems, sorry.

- Monday Nov 11th is a holiday (Veterans' Day)
  - Moving lecture of 11th to open slot:
    - Tuesday Nov 12th 6pm-9pm, Room 225-110
  - I will try and record the talk.

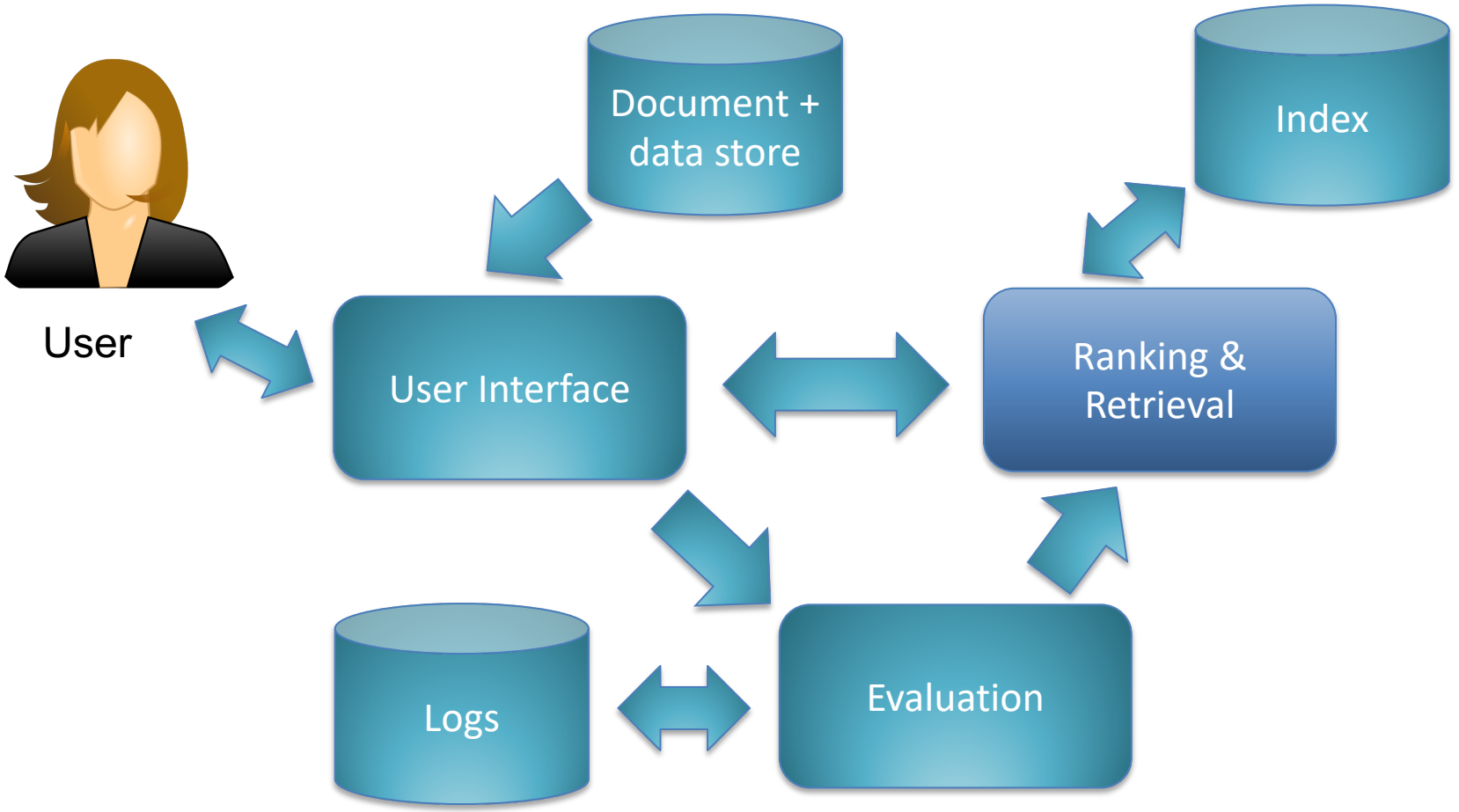
Overview*

- Retrieval Models
- Boolean vs. Ranked retrieval
- Simple Scoring
- Term frequency & Inverse Document Frequency
- Vector Space Model & Ranking
- Probabilistic Models
  - IR as classification
  - Binary Independence Model
  - BM25

* Using other sources in addition to CMS

RETRIEVAL MODELS

Search Engine: User's View (simplified)



Retrieval Models

Search: process of matching queries to documents

Retrieval Models

provide a mathematical framework for search
includes explanation of assumptions
basis of many ranking algorithms

Good models produce results that correlate with human decisions on ‘relevance’

Many different models possible

Progress in retrieval models has corresponded with improvements in effectiveness

Relevance

Complex concept that has been studied for quite some time

- Many factors to consider
- People often disagree when making relevance judgments

Retrieval models make various assumptions about relevance to simplify problem.

Two key aspects:

- e.g., *topical* vs. *user* relevance
- e.g., *binary* vs. *multi-valued* relevance

BOOLEAN VS. RANKED RETRIEVAL

Boolean Retrieval

Two possible outcomes for query processing

- TRUE and FALSE
- “exact-match” retrieval
- simplest form of ranking

Query usually specified using Boolean operators

- AND, OR, NOT
- proximity operators also used

Boolean Retrieval

Advantages

- Results are predictable, relatively easy to explain
- Many different features can be incorporated
- Efficient processing since many documents can be eliminated from search

Disadvantages

- Effectiveness depends entirely on user
- Simple queries usually don't work well
- Complex queries are difficult

Problem with Boolean search: feast or famine

- Boolean queries often result in either too many (1000s) or too few (0) results. [e.g. misspelling]
- Takes skill to come up with a query that returns a manageable number of hits.
  - AND gives too few
  - OR gives too many

“Searching by Numbers”

Sequence of queries driven by number of retrieved documents

- e.g. Search of news articles about President Lincoln
- president AND lincoln
- president AND lincoln AND NOT (automobile OR car)
- president AND lincoln AND biography AND life AND birthplace AND gettysburg AND NOT (automobile OR car)
- president AND lincoln AND (biography OR life OR birthplace OR gettysburg) AND NOT (automobile OR car)

Ranked retrieval models

Ranked retrieval: the system returns an ***ordering*** over the (top) documents in the collection for a query, rather than just an unordered set of documents

Free text queries: user's query is just one or more words in a human language, rather than using a query language with operators and expressions

In principle, 2 separate choices here

in practice, ranked retrieval associated with free text queries

Feast or famine: not a problem in ranked retrieval

- When a system produces a ranked result set, large result sets are not (as much) an issue
  - Indeed, the size of the result set is not (as much) an issue
  - We just show the top *k* (≈ 10) results
  - We don't overwhelm the user

  - Important assumption: the ranking algorithm works!

QUICK RECAP OF LOGS, Σ, Π

Sigma, Pi & Logs : A recap ..1

- You know summation $\sum_{i=1}^{n} a_i$ = $a_1 + a_2 \ldots a_n$

- Hopefully you know product $\prod_{i=1}^{n} a_i$ = $a_1 * a_2 \ldots a_n$

Note: Multiplying several small numbers can cause problems

Sigma, Pi & Logs : A recap .. 2

- Logs
  - Definition

    $\log_b a = n \rightarrow a = b^n$ e.g. $\log_{10} 100 = 2$

    log(1) = 0; log(0) is undefined. log(negative numbers?)
  - Good for squishing range of values.
    Log (1 to 100,000) → 0 to 5 !
  - Can change the base:
    $\log_a N = \log_a b \ * \ \log_b N$
    e.g. $\log_2 100 = \log_2 10 * \log_{10} 100$
  - We use a lot of $\log_2$ and $\log_{10}$

Sigma, Pi & Logs : A recap .. 3

- Logs (cont'd)
  - log(x*y) = log x + log y ; multiplication → log-addition! Cool thing!
  - [Division → ?]

- Dealing with the product of small numbers

$\log(\prod_{i=1}^{n} a_i)$

$= \log (a_1 * a_2 \ldots a_n)$

$= \log(a_1) + \log(a_2) + \ldots \log (a_n)$

$= \sum_i \log(a_i)$

SIMPLE SCORING FOR RANKED RETRIEVAL

Scoring as the basis of ranked retrieval

- We want, in order, the documents most likely to be useful to the searcher
- How can we rank-order the documents in the collection with respect to a query?
- Assign a score – say in [0, 1] – to each document
  - This score measures how well query matches document

Query-document matching scores

- Need a way of assigning a score to a query/document pair
- Consider a one-term query
  - If the query term does not occur in the document: score should be 0
  - The more frequent the query term in the document, the higher the score should be

Attempt 1: Jaccard coefficient

- Jaccard coefficient: A commonly used measure of overlap of two sets *A* and *B*

  jaccard*(A,B)* = $|A \cap B| \,/\, |A \cup B|$

  jaccard*(A,A)* = 1

  jaccard*(A,B)* = 0 if $A \cap B = 0$

Advantages

- *A* and *B* don't have to be the same size.
- Always assigns a number between 0 and 1

Jaccard coefficient: Scoring example

What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?

- Query Q: *ides of march*
- Document d1: *caesar died in march*
- Document d2: *the long march*

$\text{jaccard}(A,B) = |A \cap B| / |A \cup B|$

- *Jaccard(Q, d1) = 1/6*
- *Jaccard(Q,d2) = 1/5*

Issues with Jaccard for scoring

- Doesn't consider
  - *term frequency*
  - *rarity of terms* (more information-loaded)
- Doesn't really normalize for length

- For length normalization, can use $|A \cap B| / \sqrt{|A \cup B|}$ instead of $|A \cap B| / |A \cup B|$

TERM FREQUENCY & INVERSE DOCUMENT FREQUENCY

Bag of words model

- Bag of words model: collection of words in the document
- Representation doesn't consider the ordering of words in a document
- *John is older than Mary* and *Mary is older than John* have the same words, and the same count vectors

Term frequency tf

- Term frequency $tf_{t,d}$: number of times term *t* occurs in document *d*.
- Can we use tf when computing query-document match scores?
- Raw term frequency (count) not what we want:
  - Document with 10 occurrences of a term
    - more relevant than a document with 1 occurrence.
    - but not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Log-frequency weighting

- The log frequency weight of term t in d is

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- $\mathrm{tf}_{t,d}$ and $w_{t,d}$
  0 → 0,
  1 → 1,
  2 → 1.3,
  10 → 2,
  1000 → 4, etc.

[Why log, why 1 + ?]

Attempt 2: Log-frequency weighting

- Overlap score for a document-query pair: sum over terms *t* in both *q* and *d*:

$$\text{score} = \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$$

- The score is 0 if none of the query terms is present in the document
- Does not consider rarity

Document frequency .. 1

- Rare terms more informative than frequent terms
  - Recall stop words
- Consider query term that is rare in the collection (e.g., *defenestration*)
- A document containing this term is very likely to be relevant to the query *defenestration*

→ We want a high weight for rare terms like *defenestration*.

Document frequency .. 2

- Frequent terms less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., *high, increase, line*)
  - A document containing such a term is more likely to be relevant than a document that doesn't
  - But it's not a sure indicator of relevance

→ For frequent terms,

  - we want large positive weights for words like *high, increase, and line*
  - but lower weights than for rare terms.

- Document frequency (df) used to capture this

idf weight

- df_t is the document frequency of *t*:
  the number of documents that contain *t*
  - df_t is an inverse measure of the 'informativeness' of *t*
  - $df_t \leq N$, *(N = the total number of documents)*
- **idf** (inverse document frequency) of *t* defined by

$$\text{idf}_t = \log_{10}(N/\text{df}_t)$$

  - log (N/df_t) used instead of N/df_t to "dampen" the effect of idf
  - base of log will not matter (much)

idf example, suppose N = 1 million

term	df_t	idf_t
calpurnia	1	
animal	100	
sunday	1,000	
fly	10,000	
under	100,000	
the	1,000,000	

$$\text{idf}_t = \log_{10}(N/\text{df}_t)$$

There is only one idf value for each term t in a collection.

Effect of idf on ranking

- Assuming the word *innovative* is rarer than *company*

  For a query like [innovative company], idf weighting makes occurrences of innovative count for much more in the final document ranking than occurrences of company.

- Does idf affect ranking for one-term queries, like [Cortana]?
  - idf has no effect on ranking one term queries
  - idf affects the ranking of documents for queries with at least two terms

➔ Can we combine tf & idf ?

TF-IDF WEIGHTING

tf-idf weighting

- The tf-idf weight of a term t in a document d, in a collection of N documents, is the product of its tf weight and its idf weight.

$$W_{t,d} = tf * idf = (1 + \log(tf_{t,d})) \times \log_{10}(N/df_t)$$

- *Best known weighting scheme in information retrieval*
  - Alternatively known as: tf.idf, tf x idf
- Increases with
  - the number of occurrences within a document
  - rarity of the term in the collection

Attempt 3: Ranking Score for a document given a query

- Overlap score measure:

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

- Many variants
  - How “tf” is computed (with/without logs)
  - Whether the terms in the query are also weighted
  - etc.

VECTOR SPACE MODEL FOR SCORING

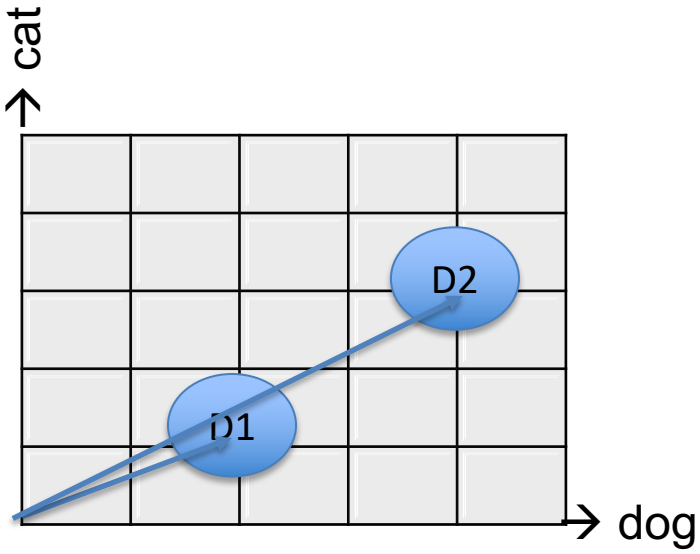
Documents as vectors .. 1

- Consider a |V|-dimensional vector space, where |V| is the size of the vocabulary V
- Each term is an axis (dimension) of the space
- Documents are points or vectors in this space

Consider documents with two dimensions (dog, cat):

D1: cat dog dog

D2: dog cat cat dog
dog cat dog



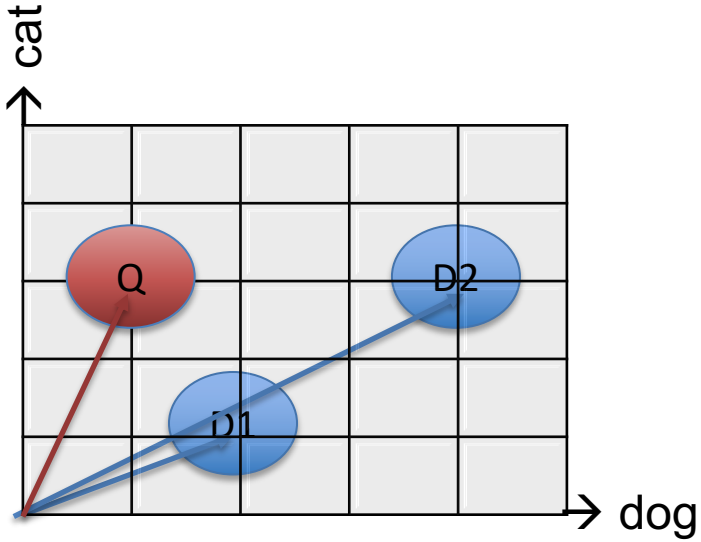
Documents as vectors .. 2

- Very high-dimensional: millions of dimensions
- Very sparse vectors - most entries are zero
- Remember: bag of words representation
  - No order of words

How do we use this for scoring?

Vector space for scoring: Treat queries also as vectors

- Key idea 1: Do the same for queries: represent them as vectors in the space
- Q: cat dog cat cat
- Now does Q match D1 better or D2?



Vector space for scoring: Rank by proximity

- Key idea 2: Rank documents according to their proximity to the query in this space
  - proximity = similarity of vectors
  - proximity ≈ inverse of distance (and dissimilarity)
- Boolean model: in or out
- Scoring model: more-relevant documents ranked higher than less-relevant documents

Formalizing vector space proximity

- How do we estimate distance between two points?

  (= distance between the end points of the two vectors)

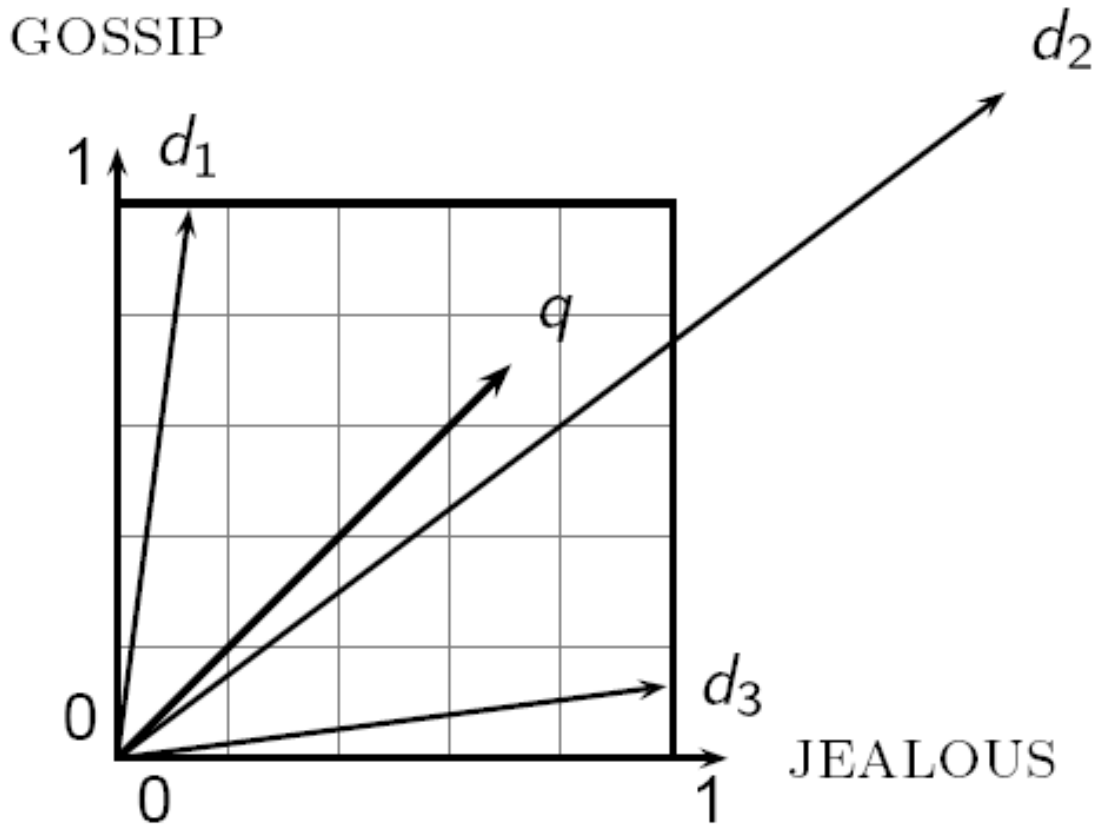
- Euclidean distance?
  - Distance between (x1, y1, …) (x2,y2,…) =

    sqrt($(x1-x2)^2 + (y1-y2)^2 + \ldots$)

- Euclidean distance is a bad idea:
  large for (similar) vectors of different lengths.

Euclidean distance? Bad idea

The Euclidean distance between $\vec{q}$ and $\vec{d_2}$ is large even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d_2}$ are very similar.



Length normalization & Unit vectors

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

- Dividing a vector by its L_2 norm makes it a unit (length) vector
- Effect on the two documents d and d’: they have identical vectors after length-normalization.
  - Long and short documents now have comparable weights

Use angle instead of distance

Thought experiment: take a document *d* and append it to itself. Call this document *d*'.

d and d' have the same content; d' is double the size

Euclidean distance between d and d' can be large

But: the angle between the two documents is 0, corresponding to maximal similarity.

Idea: Rank docs according to the angle with the query.

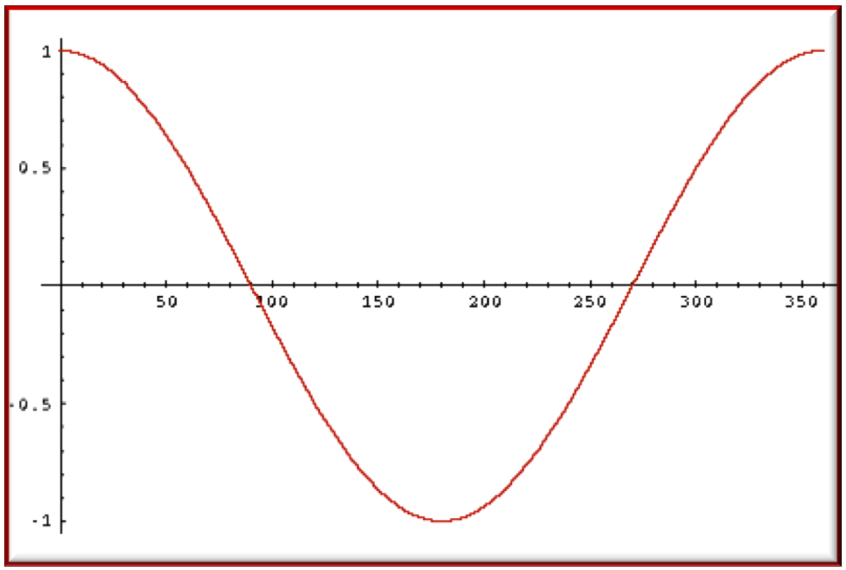
Ranking using angles & cosines

- The smaller the angle between query Q and document D_i, the more similar D_i is to Q
  - Rank documents by increasing angle subtended

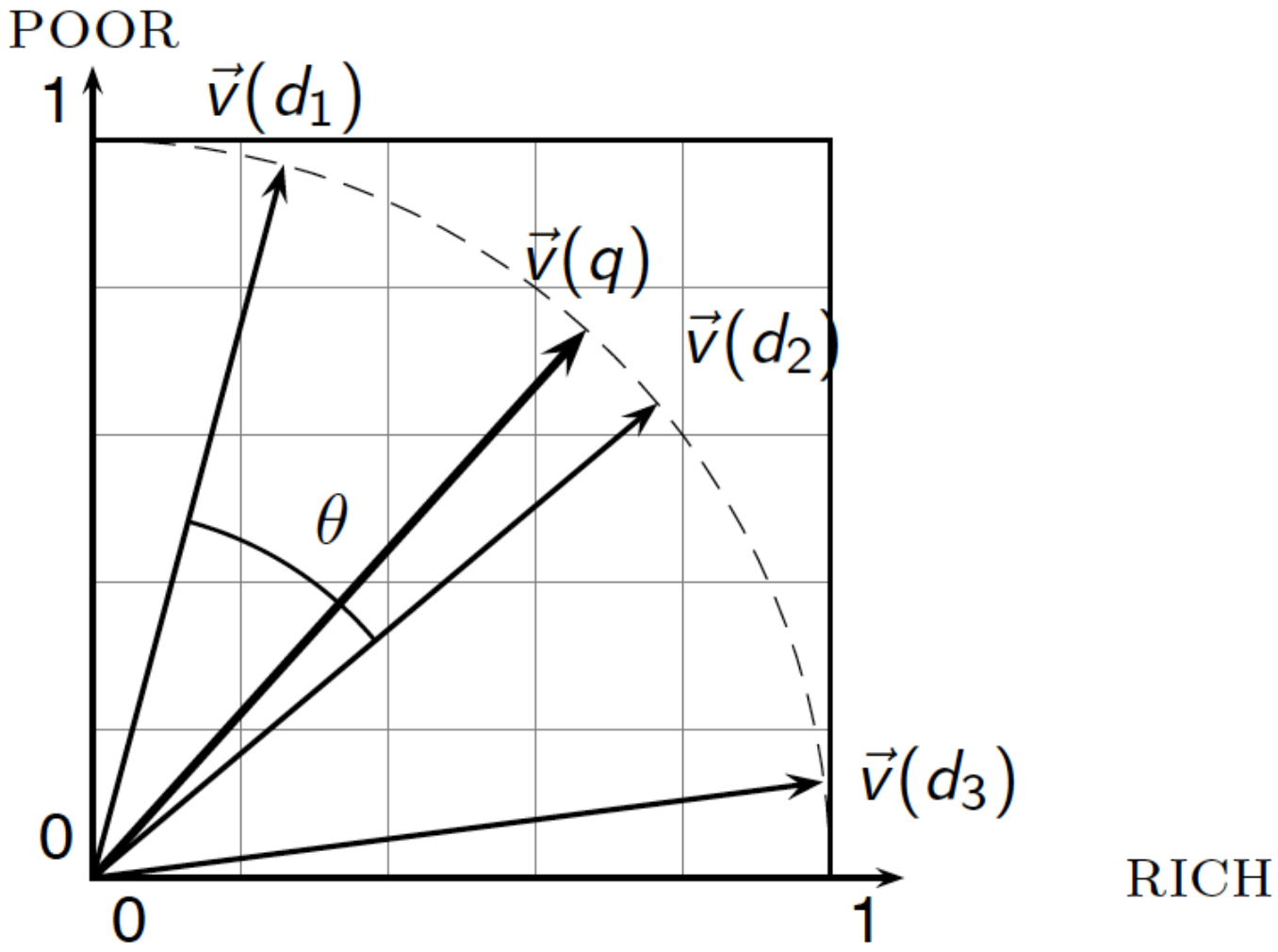
Equivalently:

- Cosine(0^o) = 1, Cosine(90^o) = 0

  Best match has cosine similarity = 1
  - Rank documents by decreasing cosine similarity



Cosine similarity illustrated



cosine(query,document)

Dot product

Unit vectors

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\bullet\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\bullet\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

q_i is the tf-idf weight of term i in the query
d_i is the tf-idf weight of term i in the document

$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

Similarity Calculation

Consider two documents D_1, D_2 and a query Q

D_1 = (0.5, 0.8, 0.3),

D_2 = (0.9, 0.4, 0.2),

Q = (1.5, 1.0, 0)

$$Cosine(D_1, Q) = \frac{(0.5 \times 1.5) + (0.8 \times 1.0)}{\sqrt{(0.5^2 + 0.8^2 + 0.3^2)(1.5^2 + 1.0^2)}}$$

$$= \frac{1.55}{\sqrt{(0.98 \times 3.25)}} = 0.87$$

$$Cosine(D_2, Q) = \frac{(0.9 \times 1.5) + (0.4 \times 1.0)}{\sqrt{(0.9^2 + 0.4^2 + 0.2^2)(1.5^2 + 1.0^2)}}$$

$$= \frac{1.75}{\sqrt{(1.01 \times 3.25)}} = 0.97$$

Cosine similarity amongst 3 documents

How similar are the novels SaS: *Sense and Sensibility* PaP: *Pride and Prejudice*, and WH: *Wuthering Heights*?

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

Note: For simplicity, assume idf = 1 for all words

3 documents example .. 1

- Log frequency weighting

  1 + log(tf)

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

term	SaS	PaP	WH
affection	1 + log(115) = 3.06	2.76	2.30
jealous	1 + log(10) = 2.00	1.85	2.04
gossip	1 + log(2) = 1.30	0	1.78
wuthering	0	0	2.58

3 documents example .. 2

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

SaS:

affection: $3.06 / \text{sqrt}(3.06^2 + 2.00^2 + 1.30^2 + 0) = 0.789$

jealous: $2.00 / \text{sqrt}(3.06^2 + 2.00^2 + 1.30^2 + 0) = 0.515$

gossip: $1.30 / \text{sqrt}(3.06^2 + 2.00^2 + 1.30^2 + 0) = 0.335$

wuthering: $0 / \text{sqrt}(3.06^2 + 2.00^2 + 1.30^2 + 0) = 0$

3 documents example .. 3

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈
0.789 × 0.832 + 0.515 × 0.555 +
0.335 × 0.0 + 0.0 × 0.0 ≈ 0.94

cos(SaS,WH) ≈ 0.79

cos(PaP,WH) ≈ 0.69

Query–Document Similarity

- Exactly like the document–document similarity we computed !

- Simplified version:

  1. Compute tf-idf vector for query
  2. for each document,

     compute tf-idf vector for document

     compute query-document similarity
  3. Choose best N

Computing cosine scores

```
CosineScore(q)
 1  float Scores[N] = 0
 2  float Length[N]
 3  for each query term t
 4  do calculate w_{t,q} and fetch postings list for t
 5      for each pair(d, tf_{t,d}) in postings list
 6      do Scores[d] += w_{t,d} × w_{t,q}
 7  Read the array Length
 8  for each d
 9  do Scores[d] = Scores[d]/Length[d]
10  return Top K components of Scores[]
```

Length: array of normalized factors (lengths) for each document

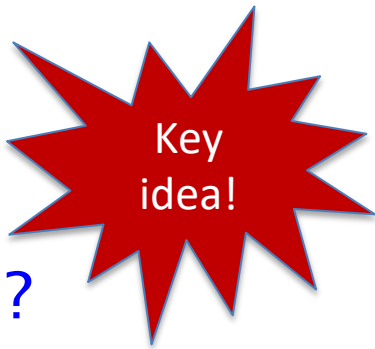
Weighting may differ in queries *vs.* documents .. 1

- Many search engines allow for different weightings for queries vs. documents
- SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq,* using the acronyms from the table on the next slide

tf-idf weighting has many variants

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

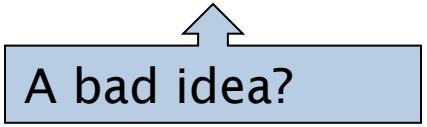
Columns headed 'n' are acronyms for weight schemes, in the form ddd.qqq
Common: **lnc.ltc**



Why is the base of the log in idf immaterial?

Weighting may differ in queries *vs.* documents .. 2

- Many search engines allow for different weightings for queries vs. documents
- SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq,* using the acronyms from the previous table
- A very standard weighting scheme is: lnc.ltc
  - Document: logarithmic tf (l as first character), no idf and cosine normalization



  - Query: logarithmic tf (l in leftmost column), log idf (t in second column), cosine normalization …

tf-idf example: lnc.ltc (ddd.qqq) .. 1

Document: *car insurance auto insurance*
Query: *best car insurance*

Term	Query					
	tf-raw	tf-weighted = (1 + log(tf))	df	idf = log(N/df)	wt = weighted-tf * idf	Nomalized wt **nq**
auto	0	0	5000	2.3	0	0
best	1	1	50000	1.3	1.3	0.34
car	1	1	10000	2.0	2.0	0.52
insurance	1	1	1000	3.0	3.0	0.78

Query: logarithmic tf (1 + log(tf)),
idf (log(N/df)),
cosine normalization

Exercise: what is *N*,
the number of docs?

tf-idf example: lnc. ltc (ddd.qqq) .. 2

Document: *car insurance auto insurance*
Query: *best car insurance*

Term	Document				
	tf-raw	tf-weighted (1 + log(tf))	idf = 1	wt = weighted-tf * idf	Nomalized wt **nd**
auto	1	1	1	1	0.52
best	0	0	1	0	0
car	1	1	1	1	0.52
insurance	2	1.3	1	1.3	0.68

Document: logarithmic tf (1 + log(tf)),
idf (= 1),
cosine normalization

tf-idf example: lnc. ltc (ddd.qqq) .. 3

Document: *car insurance auto insurance*
Query: *best car insurance*

Term	Query						Document					Product
	tf-raw	tf-wt	df	idf	wt	norm **nq**	tf-raw	tf-wt	idf	wt	norm **nd**	**nq * nd**
auto	0	0	5000	2.3	0	0	1	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0.34	0	0	1	0	0	0
car	1	1	10000	2.0	2.0	0.52	1	1	1	1	0.52	0.27
insurance	1	1	1000	3.0	3.0	0.78	2	1.3	1	1.3	0.68	0.53

wt = weighted-tf *idf

Score = 0+0+0.27+0.53 = 0.8

Redo?

Summary – vector space ranking

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity score between the query vector and each document vector
- Rank documents with respect to the query by score
- Return the top *K* (e.g., *K* = 10) to the user

How can you easily find the top K?

Vector Space Model

Advantages

- Simple computational framework for ranking
- Any similarity measure or term weighting scheme can be used

Disadvantages

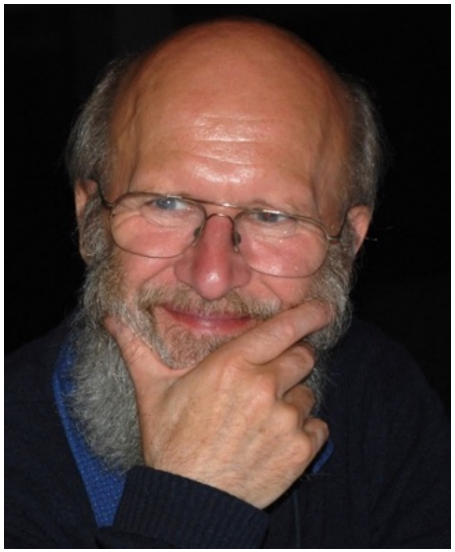
- Assumption of term independence
- No *predictions* about techniques for effective ranking

PROBABILISTIC MODELS

Probability Ranking Principle

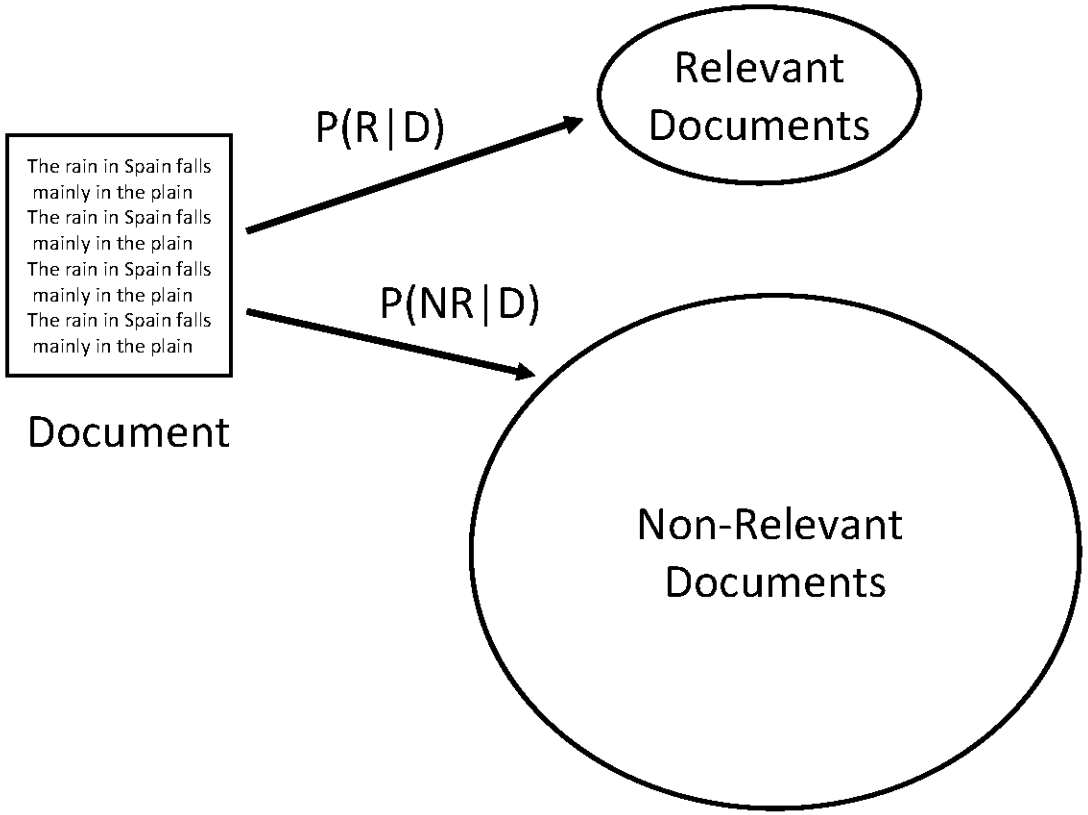
- Robertson (1977)
  - "If a reference retrieval system's response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request,
  - where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose,
  - the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data."

But: How do we estimate probabilities of relevance?



IR AS CLASSIFICATION

IR as Classification



Bayes Classifier .. 1

- Bayes Decision Rule
  - A document *D* is relevant if $P(R|D) > P(NR|D)$

P(R|D): Probability of relevance given document D
P(NR|D): Probability of non-relevance given D

- Estimating probabilities
  - use Bayes Rule $P(R|D) = P(D|R) * P(R) / P(D)$
  - Similarly $P(NR|D) = P(D|NR) * P(NR) / P(D)$

P(D|R): Probability of doc D occurring in relevant set R
P(D|NR): Probability of doc D occurring in non-relevant set NR
P(R), P(NR), P(D)

Bayes Classifier .. 1

- Bayes Decision Rule
  - A document *D* is relevant if $P(R|D) > P(NR|D)$
- Estimating probabilities
  - use Bayes Rule $P(R|D) = P(D|R)\ P(R)\ /\ P(D)$
  - Similarly $P(NR|D) = P(D|NR) * P(NR) / P(D)$

  - So, classify a document as relevant if

$$P(D|R) * P(R) / P(D) > P(D|NR) * P(NR) / P(D)$$

i.e.

$$P(D|R) * P(R) > P(D|NR) * P(NR)$$

Or

$$\frac{P(D|R)}{P(D|NR)} > \frac{P(NR)}{P(R)}$$

Left hand side is *likelihood ratio*

Estimating P(D|R)

- Assume independence

$$P(D|R) = \prod_{i=1}^{t} P(d_i|R)$$

- *Binary independence model*
  - document represented by a vector of binary features, indicating term occurrence (or non-occurrence)
  - p_i is probability that term i occurs (i.e., has value 1) in a relevant document,
  - s_i is probability that term i occurs in a non-relevant document

P(D|R) & P(D|NR)

- Assume document has terms 1, 4, 5 of five terms.
- Doc vector of binary features = (1,0,0,1,1)
- Probability of doc occurring in relevant set R:

$P(D|R) = p_1 * (1 - p_2) * (1 - p_3) * p_4 * p_5$

where p_1 = probability of term1 occurring in R, and

$(1 - p_2)$ = probability of term2 *not* occurring in R

i.e. $P(D|R) = p1 * p4 * p5 * (1-p2) * (1-p3)$

$= \prod_{i:di=1} p_i * \prod_{i:di=0}(1 - p_i)$

Similarly, $P(D|NR) = \prod_{i:di=1} s_i * \prod_{i:di=0}(1 - s_i)$

Binary Independence Model

$$\frac{P(D|R)}{P(D|NR)} = \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i}$$

with a little math manipulation

$$= \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \left(\prod_{i:d_i=1} \frac{1-s_i}{1-p_i} \cdot \prod_{i:d_i=1} \frac{1-p_i}{1-s_i} \right) \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i}$$

$$= \prod_{i:d_i=1} \frac{p_i(1-s_i)}{s_i(1-p_i)} \cdot \prod_{i} \frac{1-p_i}{1-s_i}$$

Same for all docs, can be ignored for ranking

So what do we do now to simplify this?

Take logs !!

Binary Independence Model (BIM)

- Scoring function is $\sum_{i:d_i=1} \log \frac{p_i(1-s_i)}{s_i(1-p_i)}$

Remember:
p_i is probability that term i occurs (i.e., has value 1) in a relevant document,
s_i is probability of occurrence in a non-relevant document

- Query provides information about relevant documents

Now: How do we estimate p_i and s_i?

Relevance Feedback

- Remember: User identifies relevant (and maybe non-relevant) documents in the initial result list
- Let N = total docs in a collection,
  R = number of relevant docs for a query,
  r_i = number of relevant docs containing term i ($d_i = 1$)
  n_i = number of docs containing term i
- 'Contingency table':

	Relevant
$d_i = 1$	r_i
$d_i = 0$	$R - r_i$
Total	R

BIM Scoring Function

	Relevant	Non-relevant	Total
$d_i = 1$	r_i	$n_i - r_i$	n_i
$d_i = 0$	$R - r_i$	$N - n_i - R + r_i$	$N - n_i$
Total	R	$N - R$	N

Ideally, $p_i = (r_1 / R)$; $s_i = (n_i - r_i) / (N - R)$, but to avoid 0s in logs… add 0.5 to each count, and 1 to totals, thus:

$$p_i = (r_i + 0.5)/(R + 1)$$

$$s_i = (n_i - r_i + 0.5)/(N - R + 1)$$

Substituting these in $\sum_{i:d_i=1} \log \frac{p_i(1-s_i)}{s_i(1-p_i)}$

gives us the BIM scoring function:

$$\sum_{i:d_i=q_i=1} \log \frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)}$$

How well does BIM do?

- "Binary": no tf information, so does not do well
  - tf is important!
- "independence" : words not always independent

- So not useful, but … this leads us to BM25 , which handles both tf and independence.

BM25 (Best Match formulation #25) Robertson, Walker… 1994-

- Popular and effective ranking algorithm based on binary independence model
  - adds document and query term weights

$$\sum_{i \in Q} \log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \cdot \frac{(k_1 + 1) f_i}{K + f_i} \cdot \frac{(k_2 + 1) q f_i}{k_2 + q f_i}$$

k_1, k_2 and K are parameters whose values are set empirically (or learnt)

f_i and qf_i are frequencies of term i in doc & query,

r and R set to zero if no relevance info

$$K = k_1 * ((1 - b) + b * (dl/avdl))$$

where *dl* is doc length, *avdl* = avg doc length in collection,
b is another parameter

BM25 : behaviors

Consider

$$\sum_{i \in Q} \log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \cdot \frac{(k_1+1) f_i}{K + f_i} \cdot \frac{(k_2+1) q f_i}{k_2 + q f_i}$$

- k_1 and k_2 determine tf behavior:

  if $k_1 = 0$ ➔ binary model,
  if k_1 is large, term weight increases linearly with f_i

Now consider $K = k_1((1-b) + b \cdot \frac{dl}{avdl})$

- b = 0 ➔ no length normalization,
  b = 1 ➔ full normalization

- Typical TREC value for b is 0.75

  typical TREC value of k_1 = 1.2, for very non-linear behavior (like a log function)

  typical values of k2 = 0 to 1000, since query term freq usually much smaller

BM25 Example

- Query with two terms, “president lincoln”, (*qf = 1)*
- No relevance information (*r and R are* zero)
- N = 500,000 documents
- *“president”* occurs in 40,000 documents (n_1 = 40, 000)
- *“lincoln”* occurs in 300 documents (n_2 = 300)
- “president” occurs 15 times in doc being scored (f_1 = 15)
- *“lincoln”* occurs 25 times (f_2 = 25)
- document length is 90% of the average length (*dl/avdl* = .9)
- k_1 = 1.2, b = 0.75, and k_2 = 100
- $K = 1.2 \cdot (0.25 + 0.75 \cdot 0.9) = 1.11$

BM25 Example

$$
\begin{aligned}
BM25(Q,D) \quad = \quad & \log \frac{(0+0.5)/(0-0+0.5)}{(40000-0+0.5)/(500000-40000-0+0+0.5)} \\
& \times \frac{(1.2+1)15}{1.11+15} \times \frac{(100+1)1}{100+1} \\
& + \log \frac{(0+0.5)/(0-0+0.5)}{(300-0+0.5)/(500000-300-0+0+0.5)} \\
& \times \frac{(1.2+1)25}{1.11+25} \times \frac{(100+1)1}{100+1}
\end{aligned}
$$

$$
\begin{aligned}
= \quad & \log 460000.5/40000.5 \cdot 33/16.11 \cdot 101/101 \\
& + \log 499700.5/300.5 \cdot 55/26.11 \cdot 101/101 \\
= \quad & 2.44 \cdot 2.05 \cdot 1 + 7.42 \cdot 2.11 \cdot 1 \\
= \quad & 5.00 + 15.66 = 20.66
\end{aligned}
$$

BM25 Example: Effect of term frequencies

Frequency of "president"	Frequency of "lincoln"	BM25 score
15	25	20.66
15	1	12.74
15	0	5.00
1	25	18.2
0	25	15.66

- With no relevance info, like idf weight
- "lincoln" has lower df, higher idf; even tf=1 is big
- Note also high score for doc with only "lincoln", no "president"

BM25 Summary

Effective ranking algorithm

- derived from “IR as classification” model

Focuses on topical relevance (doc “on topic”)

Assumes relevance is binary

Note: In implementation:

- Weights can be tuned to document collection
- Some calculation of BM25 weights can be computed at index-time

Summary

- Defined retrieval models
- Compared Booleans and ranked retrieval
- Considered a simple scoring scheme
- Studied term frequency and inverse doc frequency, and went on to tf-idf weighting
- Understood the Vector Space Model and how it can be used to score documents
- Looked at Probabilistic Retrieval Models, with a focus on two: binary independence model and BM25
- Next week: **Retrieval Models 2**

Readings

- Croft, Metzler & Strohman (CMS), Sections 7.1, 7.2

Optional:

- Manning, Raghavan & Schütze (MRS), IIR Sections 6.2 – 6.4.3

Quiz 7

Assignment 3

Questions?